

# **rcrypt packer**

by Feiad Mohammed

## **rcrypt features**

- timelock puzzle for delayed execution
- anti virtualization (vbox)
- fake imports
- supports EXE and DLL files

## **rcrypt purpose**

rcrypt is a crypter (a type of packer) and its purpose is similar to others of its kind. To protect intellectual property. If you're familiar with commercial software, music, movies, games, etc you are familiar with this type of protection. If you create some fancy new optimization algorithm and want to profit from it you do not want someone to simply reverse engineer your binary and make use of your hard work. When commercial entities use this kind of binary protection it is called DRM (Digital Rights Management). Rcrypt offers users the ability to encrypt their binaries to make reverse engineering harder. No solution is absolute and rcrypt is no exception. It is a proof of concept and nothing more. I make no guarantee that your binary will even function after packing so your mileage may vary. Just be aware that I do not condone malicious use of rcrypt as it is designed for educational purposes only.

## **rcrypt features**

The timelock puzzle functionality causes delay in execution to avoid virtualized scanning techniques employed by Antivirus products. In rcrypt a timelock puzzle is implemented by the brute forcing of the crypto key used to encrypt the binary with varying difficulty specified by optional parameter *-complex hexvalue*.

The encryption algorithm used in rcrypt is XXTEA [1] also known as Corrected Block TEA. I am a fan of the Tiny Encryption Algorithm as it can be implemented entirely with CPU registers with no need for stack variables. It is a very efficient algorithm. Seeing as XTEA supersedes it, followed immediately by XXTEA which supersedes both I decided to use XXTEA. It is indeed the ultimate of TEAs.

Binaries with no or very few imports are highly suspicious. rcrypt will add fake imports to packed binaries. The imports I've chosen are based on the default imports created by a hello world Visual Studio program built in release mode.

Anti virtualization is currently limited to Virtualbox. I plan to add support for others as time permits. VMWare is far too ubiquitous to have an anti for as many systems actually run on it.

**Additional features not explicitly mentioned include the following:**

***anti debugger functionality***

obviously designed to detect debuggers to make reversing more difficult.

***polymorphic self destructing stub***

the unpacking stub kills itself after unpacking completes. To support DLLs there is a small portion that remains after the self destruct feature.

## **Anti-virus Scanning Methodologies and Malware**

Antivirus products have been made a fool of for a very long time. Signature based scanning, no matter how well done, cannot deal with packed executables as they completely mask the byte patterns used to detect malware. This is why sandbox scanning methods are now the main method used by the best products out there (ie: Kaspersky comes to mind). Using these emulated environments to allow binaries to execute defeats the binary obfuscation offered by packers as all packers must unpack in memory. Virtualization has become ubiquitous enough to allow isolated environments to be created for every single EXE or DLL file that an antivirus scanner might want to scan.

However these full blown virtualized environments have overhead and cannot be run throughout the entire lifetime of the target binary executable. If a user had to wait for an executable to fully execute they might be waiting a long time. Especially since most binaries are graphical user interfaces or otherwise programs awaiting user input. Imagine this happening for countless binaries at any given time and having to wait for a double clicked program to start but only after it has been virtualized in its entirety to determine whether or not it was malicious. This is why these scanning techniques can only be employed for a short amount of time. Due to the nature of malicious binaries a few seconds is generally all they require to unpack and modify/exfiltrate/beacon which will trigger detection. This makes even a few seconds of virtualized based scanning worthwhile.

Pausing for a short amount of time is a very well known method of bypassing this type of scanning technique and has been in use for a long time. Just imagine a function like the following:

```
evilFunction  
{  
    Sleep(60000);  
    AllTheBadThings();  
}
```

Although using Sleep() is so common it is now ignored by almost every scanner out there the point is still there. If you can cause execution to delay for a while then you can bypass such scanning methods.

Ok so are there other ways to delay execution?

## **Timelock Puzzle**

One of the more interesting ways to produce a delay in execution is to use cryptographic functionality to somewhat predictably cause execution delay. Various cryptographic computations have been proposed and do a great job in doing pointless work to cause delay.

## **Pointless Computations and Avoidability**

A human analyst can detect useless computations if they can determine pointless functionality and remove it from the executable code as it is not a necessary part of the binary.

## **Timelock Puzzle in rcrypt**

I liked the general idea behind timelock puzzles and thought that making use of cryptographic functionality while brute forcing a crypto key would be sufficient to cause delay. In rcrypt when a binary is encrypted a randomly generated key is used. Part of the packer's in memory routine is now to brute force this key which will facilitate delay and require the brute forcing as a necessary part of the process.

## **Example rcrypt packed malware detection**

After scanning a known malware sample packed with rcrypt against a fully updated KAV I determined that this was sufficient to bypass a typical sandbox used by Antivirus scanning engines. Rcrypt has an optional parameter `-complex 0x1234` users can use to tweak the difficulty of the brute forcing routine to increase/decrease delay.

For demonstration I will take a very well known malware released ten years ago that is detected by every Antivirus product and pack it.

```
Command Prompt

rcrypt v1.2 c0ded by rage [Feiad Mohammed]

This program was written for educational purposes only.
I will not be held liable for any damage caused by the <mis>use
of this tool.

This tool was designed for peace, love, and rainbows. Any illegal
use of this program is strictly prohibited and your sole responsibility.

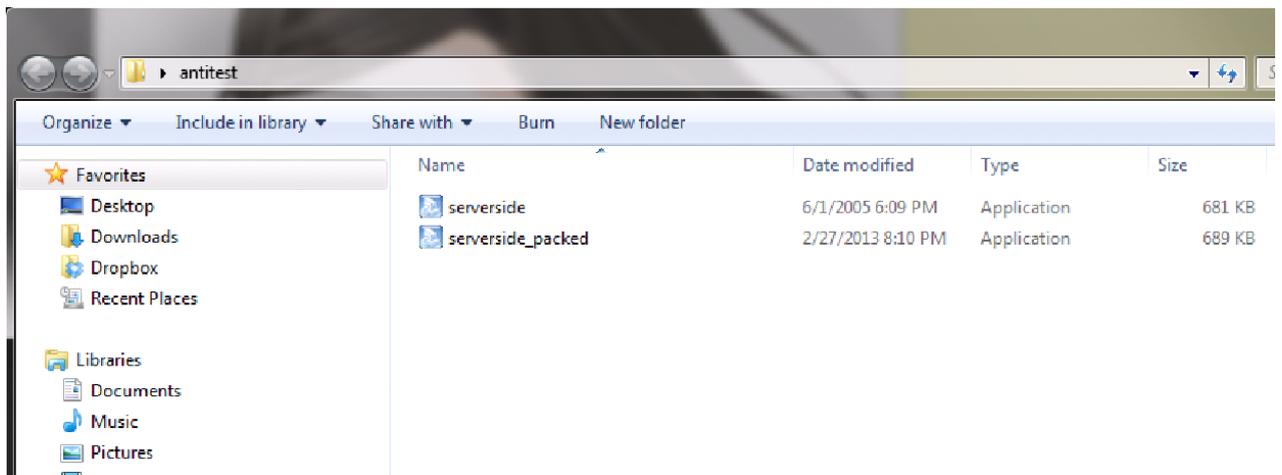
If you dont agree delete it and get back to your regularly scheduled
program.

*/
opening file: serverside_packed.exe
crypto brute force complexity has been set to value: 0x1fff (8191)
exe detected...
new crypto key generated..
writing unpackstub..
encoding..
adding imports..
wrote kernel imports...current written: 35e
wrote user imports...current written: 379
copying data..
unpackstub written..

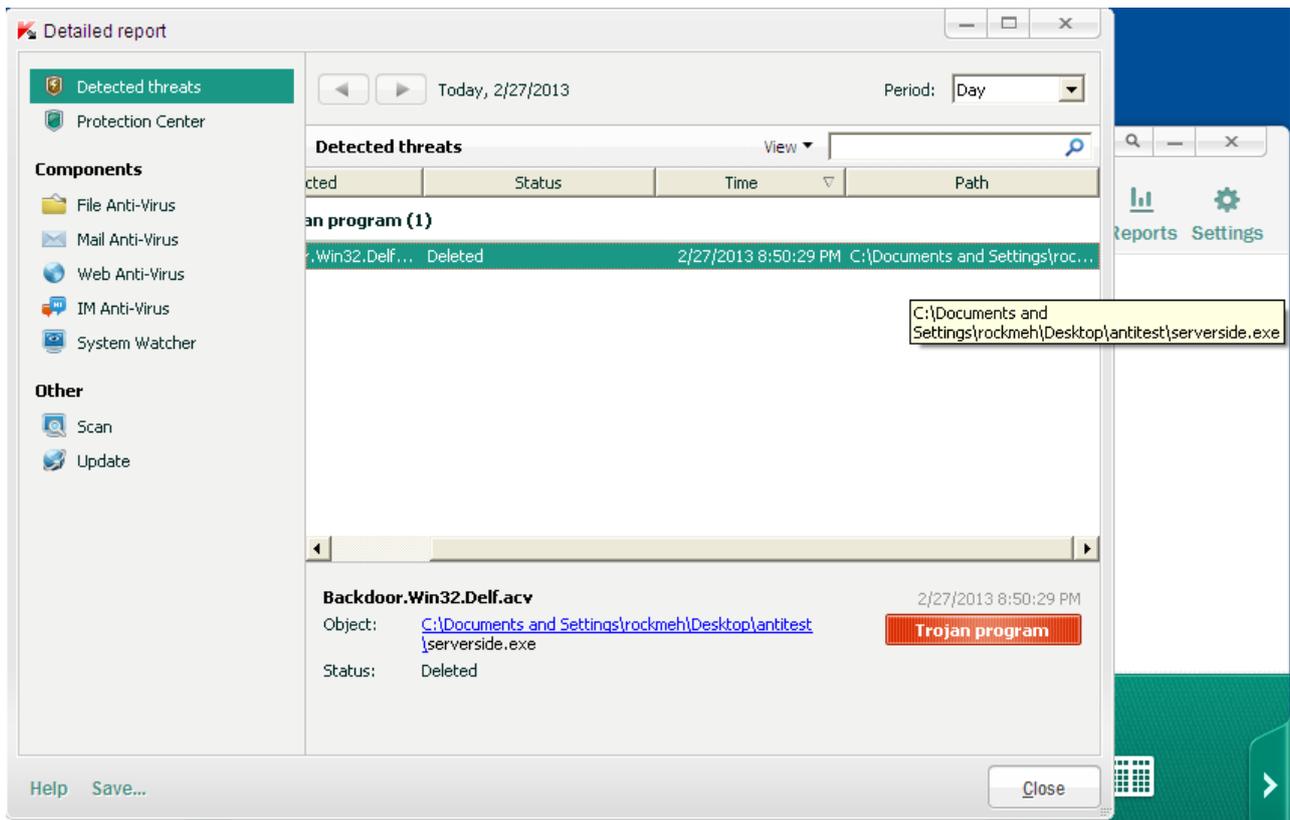
C:\Users\RaGe\Desktop\rcryptv12>
```

The malware sample is called Insurrection and has a client and server component. The server file is named serverside.exe and is copied to the rcrypt directory and packed.

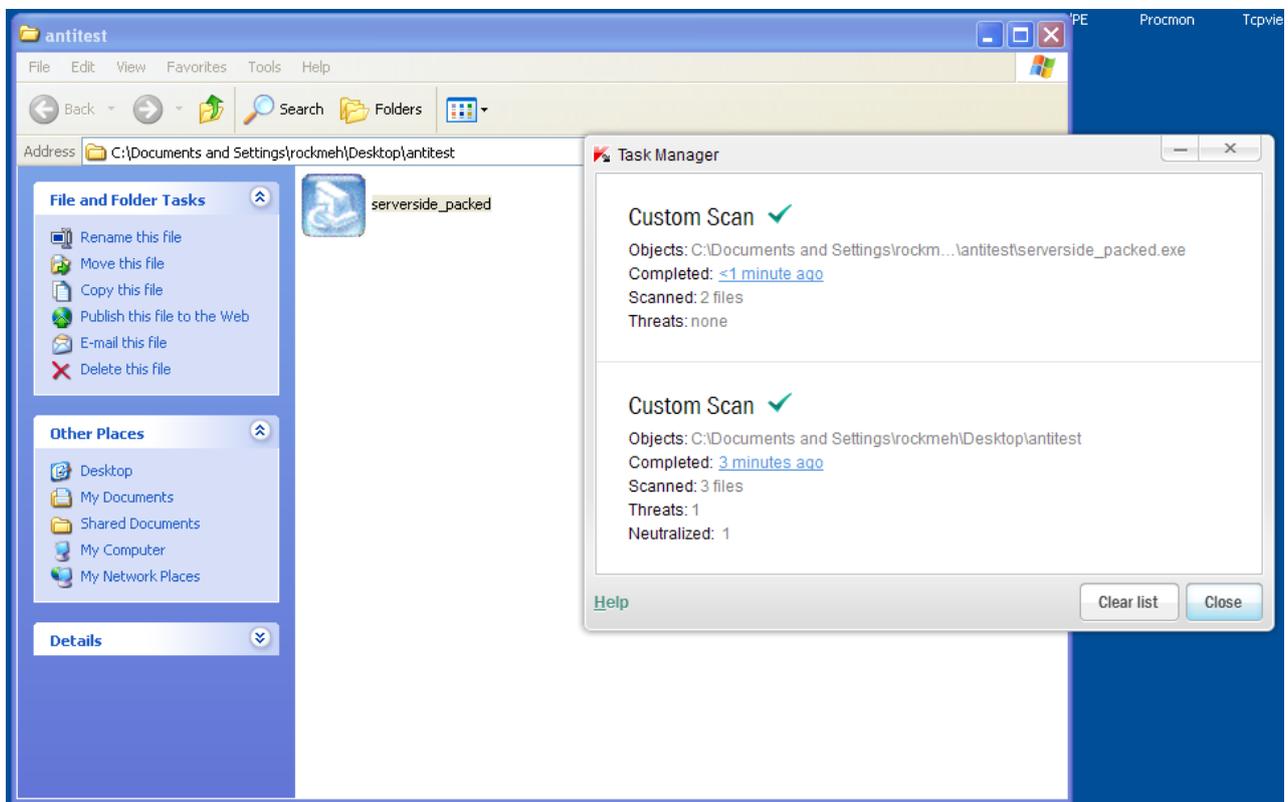
Below you can see the packed and unpacked binaries in a folder.



KAV is installed in a VM and updated and detects the unpacked binary immediately but does not detect the packed version.



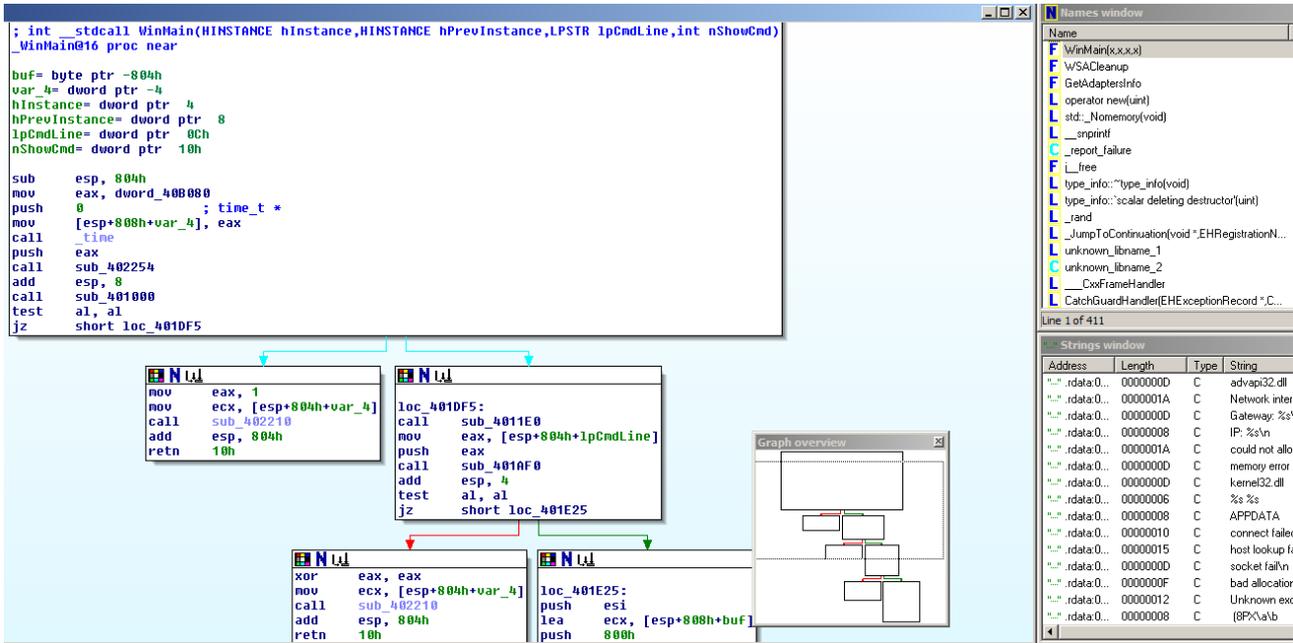
Unpacked detection.



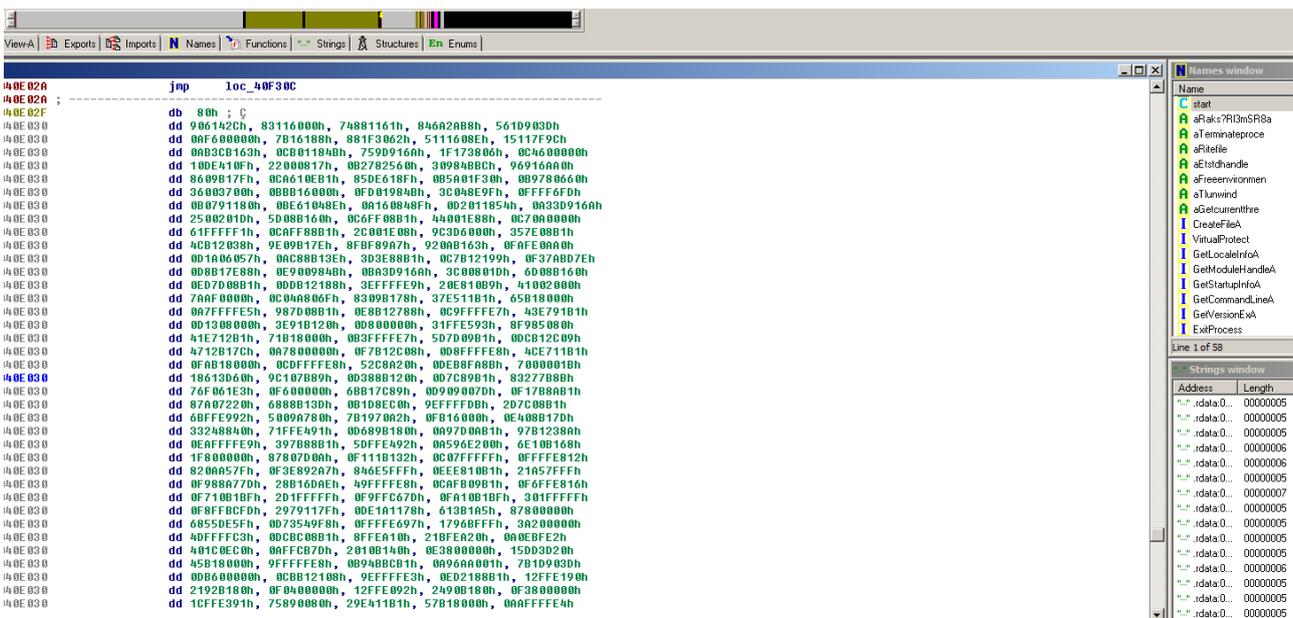
This shows that rcrpt delays execution long enough to timeout KAV's sandbox scanning technology.

## rcrypt Reversing in IDA Pro

Because the sample is packed IDA cannot determine original functionality or even identify functions or other known code. Below is an image of IDA Pro after loading the unpacked sample.

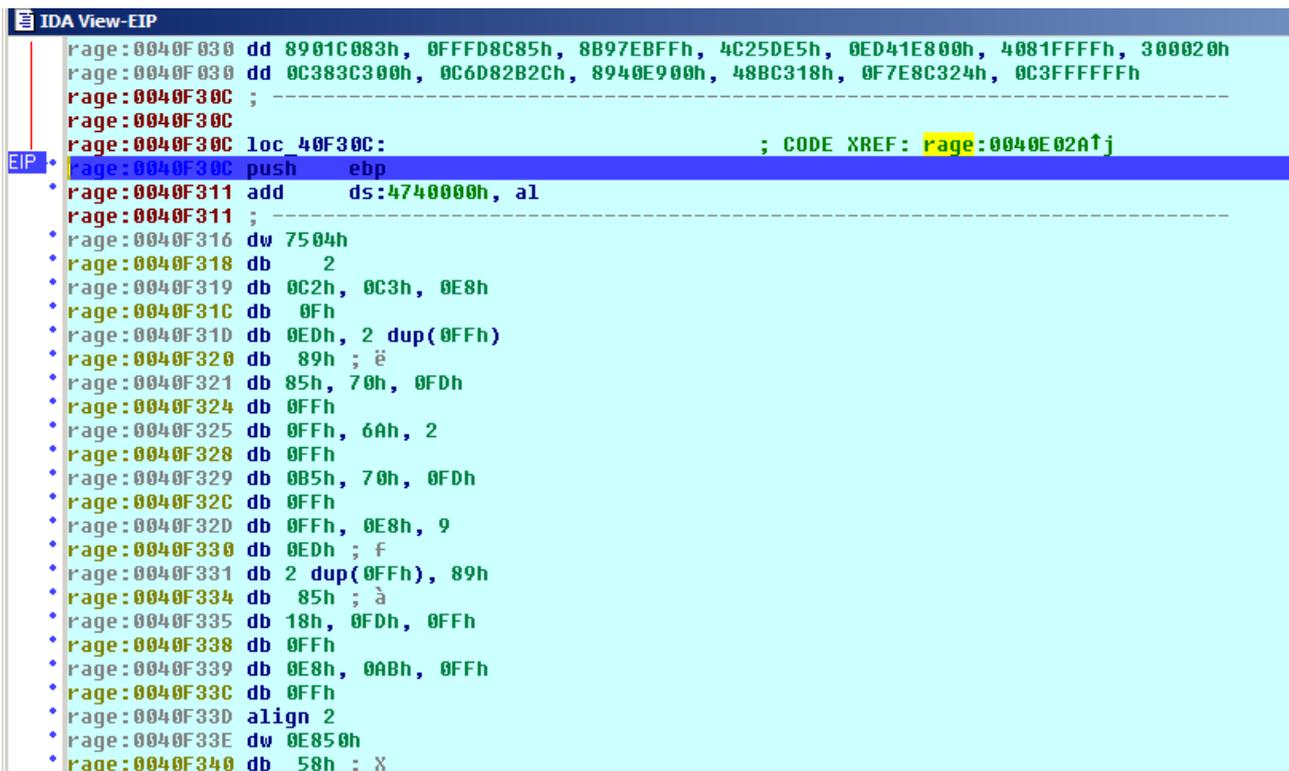


You can see IDA Pro has been able to produce a nice control flow graph. Now here is an image of IDA Pro after loading the packed version.



As you can see you get quite a bit of garbage. Also note that the pretty control flow graphs newbie reversers tend to be over dependent upon are no longer produced by IDA. Static analysis is useless on packed binaries. Luckily we can use the built in debugger and step through code and IDA will update the view based on new information gleaned.

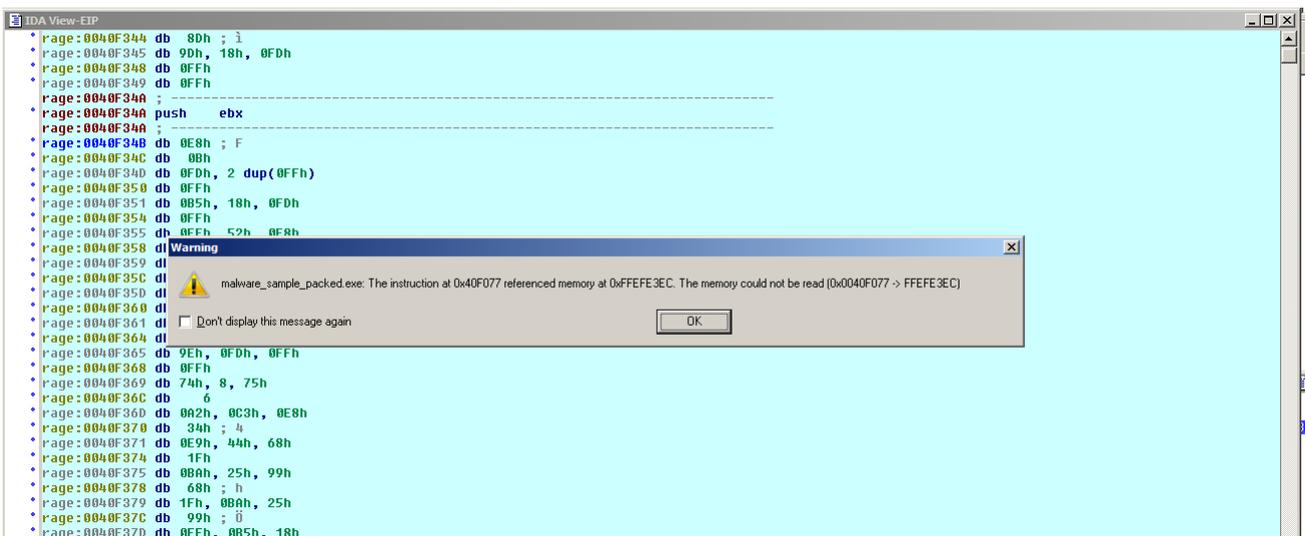
I did just that and produced the following results.



```
IDA View-EIP
rage:0040F030 dd 8901C083h, 0FFFD8C85h, 8B97EBFFh, 4C25DE5h, 0ED41E800h, 4081FFFFh, 300020h
rage:0040F030 dd 0C383C300h, 0C6D82B2Ch, 8940E900h, 48BC318h, 0F7E8C324h, 0C3FFFFFFFh
rage:0040F30C ; -----
rage:0040F30C
rage:0040F30C loc_40F30C: ; CODE XREF: rage:0040E02A↑j
EIP *rage:0040F30C push ebp
*rage:0040F311 add ds:4740000h, a1
rage:0040F311 ; -----
*rage:0040F316 dw 7504h
*rage:0040F318 db 2
*rage:0040F319 db 0C2h, 0C3h, 0E8h
*rage:0040F31C db 0Fh
*rage:0040F31D db 0EDh, 2 dup(0FFh)
*rage:0040F320 db 89h ; ä
*rage:0040F321 db 85h, 70h, 0FDh
*rage:0040F324 db 0FFh
*rage:0040F325 db 0FFh, 6Ah, 2
*rage:0040F328 db 0FFh
*rage:0040F329 db 0B5h, 70h, 0FDh
*rage:0040F32C db 0FFh
*rage:0040F32D db 0FFh, 0E8h, 9
*rage:0040F330 db 0EDh ; f
*rage:0040F331 db 2 dup(0FFh), 89h
*rage:0040F334 db 85h ; à
*rage:0040F335 db 18h, 0FDh, 0FFh
*rage:0040F338 db 0FFh
*rage:0040F339 db 0E8h, 0ABh, 0FFh
*rage:0040F33C db 0FFh
*rage:0040F33D align 2
*rage:0040F33E dw 0E850h
*rage:0040F340 db 58h ; X
```

Here I am continuing to step through code and there is still no graph or easily identifiable code constructs.

And then finally....



```
IDA View-EIP
*rage:0040F344 db 8Dh ; ï
*rage:0040F345 db 9Dh, 18h, 0FDh
*rage:0040F348 db 0FFh
*rage:0040F349 db 0FFh
rage:0040F34A ; -----
*rage:0040F34A push ebx
rage:0040F34A ; -----
*rage:0040F34B db 0E8h ; F
*rage:0040F34C db 0Bh
*rage:0040F34D db 0FDh, 2 dup(0FFh)
*rage:0040F350 db 0FFh
*rage:0040F351 db 0B5h, 18h, 0FDh
*rage:0040F354 db 0FFh
*rage:0040F355 db 0EFh, 52h, 0FAh
*rage:0040F358 di Warning
*rage:0040F359 di
*rage:0040F35C di
*rage:0040F35D di
*rage:0040F360 di
*rage:0040F361 di
*rage:0040F364 di
*rage:0040F365 db 9Eh, 0FDh, 0FFh
*rage:0040F368 db 0FFh
*rage:0040F369 db 74h, 8, 75h
*rage:0040F36C db 6
*rage:0040F36D db 0A2h, 0C3h, 0E8h
*rage:0040F370 db 34h ; ù
*rage:0040F371 db 0E9h, 44h, 68h
*rage:0040F374 db 1Fh
*rage:0040F375 db 0BAh, 25h, 99h
*rage:0040F378 db 68h ; h
*rage:0040F379 db 1Fh, 0BAh, 25h
*rage:0040F37C db 99h ; ð
*rage:0040F37D db 0FFh, 0B5h, 18h
```

Warning: malware\_sample\_packed.exe: The instruction at 0x40F077 referenced memory at 0xFFFE3EC. The memory could not be read (0x0040F077 -> FFE3EC)

Don't display this message again

OK

IDA Pro crashes as it is unable to continue execution. As you can see this is what IDA Pro looks like when stepping through rcrypt packed executable code. There are many things here designed to trip up analysis and frustrate reverse engineering.

## **Detecting Packed Malware**

To detect packed malware Antivirus products employ unpacking functionality for each supported packer to allow scanning of the original binary. One would simply need to be created for rcrypt. Of course this kind of approach does not scale as there are many packers out there.

## **Full Blown Virtualization**

Automated analysis tools are a dime a dozen these days and will allow a user to upload binaries and get reports back from their virtualized analysis engines. These virtual environments run executables for arbitrary amounts of time so the delay in execution does not deter such analysis. Starting with rcrypt 1.3 there is an optional parameter "*-trick1*" that will attempt to detect the most typical sandbox environments and cause early termination. This is an option that shows that overreliance on simple automated solutions can be very misleading. Never rely solely on your tools as any tool can be fooled if someone wants to do so.

## **Conclusion**

It is clear that the arms race between detecting malware and evading detection systems has produced some interesting new products and techniques. However no single solution solves everything and a layered security model is needed. There are always ways around detection techniques and evading techniques and all one can do is raise the bar as high as possible to lower the chance of falling prey to a successful attack.

My website: <http://www.0xrage.com>

## References

[1] <http://en.wikipedia.org/wiki/XXTEA>